CS671 Parallel Programming in the Many-Core Era

Polyhedral Framework for Compilation: Polyhedral Model Representation, Data Dependence Analysis, Scheduling and Data Locality Optimizations

December 10, 2013
Image and Preimage

Definition: Image

The image of a polyhedron \( \mathcal{P} \in \mathbb{Z}^n \) by an affine function \( f: \mathbb{Z}^n \rightarrow \mathbb{Z}^m \) is:

\[
\mathcal{P}' = \{ f(\bar{x}) \in \mathbb{Z}^m | \bar{x} \in \mathcal{P} \}
\]

Definition: Pre-Image

The preimage of a polyhedron \( \mathcal{P} \in \mathbb{Z}^n \) by an affine function \( f \) is:

\[
\mathcal{P}' = \{ \bar{x} \in \mathbb{Z}^n | f(\bar{x}) \in \mathcal{P} \}
\]

We have \( \text{Image}(f^{-1}, \mathcal{P}) = \text{Preimage}(f, \mathcal{P}) \) if \( f \) is invertible.
Data Dependence

**Definition: Bernstein conditions**
Given two references, there exists a dependence between them if the three following conditions hold:

- They reference the same array (cell)
- One of them is a write
- The two associated statements are executed

**Review: different types of dependences:**

- **RAW (Read-After-Write):** parallelization analysis
- **WAR (Write-After-Read):** parallelization analysis
- **WAW (Write-After-Write):** parallelization analysis
- **RAR (Read-After-Read):** data locality/reuse computations
Purpose of Dependence Analysis

Preserve the semantics:

\[
\text{Semantics is preserved if the dependence are preserved}
\]

- Statement as unit: a reference to an \textbf{array}
- Statement instance as unit: a reference to an \textbf{array cell}
An Intuitive Dependence Test Algorithm

Given two references $a$ and $b$ to the same array:

- **Write and read sets for $a$**
  - $\mathcal{W}_a : \text{Image}(f_a, D_a)$ if $a$ is a write
  - $\mathcal{R}_a : \text{Image}(f_a, D_a)$ if $a$ is a read

- **Write and read sets for $b**
  - $\mathcal{W}_b : \text{Image}(f_b, D_b)$ if $b$ is a write
  - $\mathcal{R}_b : \text{Image}(f_b, D_b)$ if $b$ is a read

If $\mathcal{W}_a \cap \mathcal{R}_b \neq \emptyset \lor \mathcal{W}_a \cap \mathcal{W}_b \neq \emptyset \lor \mathcal{R}_a \cap \mathcal{W}_b \neq \emptyset$ then

$$a \delta b$$
A Dependence Test Algorithm

- Create the Data Dependence Graph (DDG)
  Create one node for every statement

- For all pairs $a, b$ of distinct references,
  (i) Compute $\mathcal{W}_a, \mathcal{R}_a, \mathcal{W}_b, \mathcal{R}_b$
  (ii) If $\mathcal{W}_a \cap \mathcal{R}_b \neq \emptyset \lor \mathcal{W}_a \cap \mathcal{W}_b \neq \emptyset \lor \mathcal{R}_a \cap \mathcal{W}_b \neq \emptyset$ then
    Add an edge between the statement with the reference $a$
    and the statement with the reference $b$
Statement Instances in Dependence

- **Objective:**
  Find the set of instances in dependence.

- **Example:**
  \( \text{preimage}(f_a, \mathcal{W}_a \cap \mathcal{R}_b) \) gives the set of statement instances that has RAW dependence from \( \mathcal{W}_a \) to \( \mathcal{R}_b \).
Different Dependence Types for Statement Instances

Examples:

▶ Uniform dependence
   *The distance between two dependent iterations is a constant*
   Example: $i \rightarrow i + 1$

▶ Non-uniform dependence
   *The distance between two dependent iterations varies*
   Example: $i \rightarrow i + j$

▶ Parametric dependence
   At least a parameter is involved in the dependence relation
   Example: $i \rightarrow i + N$
Approach for Finding Dependent Statement Instances

Objective: Obtain the set of statements in dependence

Solution:
- **Dependence polyhedron**: check if sets of dependent instances exist
Definition:
A statement $S$ depends on a statement $R$ (written $R \rightarrow S$) if there exists an operation $S(\vec{x}_S)$, $R(\vec{x}_R)$, a memory location $m$, and $\vec{x}_S, \vec{x}_R$ are respectively two iteration instances such that:

- $S(\vec{x}_S)$ and $R(\vec{x}_R)$ refer to the same memory location $m$, and at least one of them writes to that location
- $\vec{x}_S$ and $\vec{x}_R$ belongs to the iteration domain of $R$ and $S$,
- In the original sequential order, $R(\vec{x}_R)$ is executed before $S(\vec{x}_S)$. 
Define the loop:
\[
\begin{align*}
\text{do } x_1 &= L_1, U_1 \\
&\text{ for } x_2 = L_2, U_2 \\
&\quad \ldots \\
&\quad \text{ for } x_n = L_n, U_n \\
&\quad \ldots \\
&\quad \text{R: Data}[F_R \times \vec{x} + a_R] = \ldots \\
&\quad \text{S: } \ldots = \text{Data}[F_S \times \vec{x} + a_S] \\
&\quad \ldots \\
&\text{ endfor} \\
&\text{ end for} \\
&\text{ end for}
\end{align*}
\]
Dependence of Statement Instances III

Linear Constraints Imposed by Last Three Conditions

- S(\vec{x}_S) and R(\vec{x}_R) reference the same memory location:
  \[ F_s \vec{x}_S + a_S = F_R \vec{x}_R + a_R. \]
- \vec{x}_R and \vec{x}_S within loop iteration domains:
  \[ A_S \vec{x}_S + c_S \geq 0 \] and \[ A_R \vec{x}_R + c_R \geq 0. \]
- Precedence order: R(\vec{x}_R) happens before S(\vec{x}_S) for a particular dependence loop level \( L \) where the dependence happens,

\[
\begin{align*}
\text{for } i < L: & \quad \vec{x}_{R,i} = \vec{x}_{S,i} \\
\text{for } i = L: & \quad \vec{x}_{R,L} \leq \vec{x}_{S,L}
\end{align*}
\]

These linear equalities and inequalities can be rewritten as:

\[ P_{I,s} \vec{x}_S - P_{I,R} \vec{x}_R + b \geq 0. \]
The dependence polyhedron for $\mathcal{R} \rightarrow \mathcal{S}$ at a given level $l$ and for a given pair of references $f_R, f_S$ is described as [Feautrier/Bastoul]:

$$D_{R,S,f_R,f_S,l} :$$

$$D \left( \begin{array}{c} \vec{x}_S \\ \vec{x}_R \end{array} \right) + d = \begin{bmatrix} F_S & -F_R \\ A_S & 0 \\ 0 & A_R \\ P_S & -P_R \end{bmatrix} \left( \begin{array}{c} \vec{x}_S \\ \vec{x}_R \end{array} \right) + \begin{bmatrix} a_S - a_R \\ c_S \\ c_R \\ b \end{bmatrix} \geq 0$$
Polyhedral Representation of Programs

Static Control Part

- Loops have affine control only
- Iteration domain: integer polyhedra ($\vec{x}_S$)
- Memory accesses: static ref. $\rightarrow$ affine functions of $\vec{x}_S$ and $\vec{p}$
- Data dependences between $S_1$ and $S_2$:
  A subset of the Cartesian product of $D_{S_1}$ and $D_{S_2}$
Examples: static memory access and data dependence

The example loop

```
for ( i = 1; i <= 3; ++i ) {
  S1: a[i] = i;
  for ( j = 1; j <= 3; ++j )
    S2: b[j] = (b[j] + a[i])/2;
}
```

\[ D_{S1\delta S2,<a[i_{S1}],a[i_{S2}]>} : \begin{bmatrix}
1 & -1 & 0 & 0 \\
1 & 0 & 0 & -1 \\
-1 & 0 & 0 & 3 \\
0 & 1 & 0 & -1 \\
0 & -1 & 0 & 3 \\
0 & 0 & 1 & -1 \\
0 & 0 & -1 & 3 \\
\end{bmatrix} \begin{bmatrix}
i_{S1} \\
i_{S2} \\
j_{S2} \\
1 \\
\end{bmatrix} = \vec{0} \geq \vec{0} \]
Examples: static memory access and data dependence

The example loop

```c
for ( i = 1; i <= 3; ++i ) {
    S1: a[i] = i;
    for ( j = 1; j <= 3; ++j )
        S2: b[j] = (b[j] + a[i])/2;
}
```

\[ \mathcal{D}_{s_2 \delta s_2, <b[i_{s_2'}], b[j_{s_2}]>, 1} : \]

\[
\begin{bmatrix}
  0 & 1 & 0 & 1 & 0 \\
  1 & 0 & 0 & 0 & -1 \\
  -1 & 0 & 0 & 0 & 3 \\
  0 & 1 & 0 & 0 & -1 \\
  0 & -1 & 0 & 0 & 3 \\
  0 & 0 & 1 & 0 & -1 \\
  0 & 0 & -1 & 0 & 3 \\
  0 & 0 & 0 & 1 & -1 \\
  0 & 0 & 0 & -1 & 3 \\
  -1 & 0 & 1 & 0 & -1 \\
\end{bmatrix}
\begin{bmatrix}
  i_{s_2'} \\
  j_{s_2'} \\
  i_{s_2} \\
  j_{s_2} \\
  1 \\
\end{bmatrix}
\begin{bmatrix}
  \vec{0} \\
  \geq \vec{0}
\end{bmatrix}
\]
Dependence polyhedra algorithm

Step by step:

Initialize DDG graph with one node for every statement
Dependence polyhedra algorithm

Step by step:

Initialize DDG graph with one node for every statement

For each pair of statements $R, S$
Dependence polyhedra algorithm

Step by step:

 Initialize DDG graph with one node for every statement
   For each pair of statements $R, S$
     For each pair of references $f_R, f_S$
Dependence polyhedra algorithm

Step by step:

Initialize DDG graph with one node for every statement
For each pair of statements $R, S$
For each pair of references $f_R, f_S$
For each loop level $l$: min_depth to common_depth

Build dependence polyhedron $D_{R,S,f_R,f_S,l}$
If it is not empty, then get the dependence type
Add edge $(R, S, \{l, D_{R,S,f_R,f_S,l}, type\})$
Dependence polyhedra algorithm

Step by step:

Initialize DDG graph with one node for every statement

For each pair of statements $R, S$

For each pair of references $f_R, f_S$

For each loop level $l$: min_depth to common_depth

Build dependence polyhedron $\mathcal{D}_{R,S,f_R,f_S,l}$
Dependence polyhedra algorithm

Step by step:

Initialize DDG graph with one node for every statement

For each pair of statements $R, S$

For each pair of references $f_R, f_S$

For each loop level $l$: min_depth to common_depth

Build dependence polyhedron $D_{R,S,f_R,f_S,l}$

If it is not empty, then get the dependence type
Dependence polyhedra algorithm

Step by step:

Initialize DDG graph with one node for every statement

For each pair of statements $R, S$

For each pair of references $f_R, f_S$

For each loop level $l$: $\text{min}\_\text{depth}$ to $\text{common}\_\text{depth}$

Build dependence polyhedron $\mathcal{D}_{R,S,f_R,f_S,l}$

If it is not empty, then get the dependence type

Add edge$(R,S,\{l, \mathcal{D}_{R,S,f_R,f_S,l}, \text{type}\})$
Check if the dependence polyhedron is empty using PiPLib:

- Given a linear system describing a polyhedron:
- It is written as for PolyLib matrix format:

\[ D_{S_2 \delta S_2, <b[j_{s_1}], b[j_{s_2}]>, 1} : \]

\[
\begin{bmatrix}
0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & -1 \\
-1 & 0 & 0 & 0 & 3 \\
0 & 1 & 0 & 0 & -1 \\
0 & -1 & 0 & 0 & 3 \\
0 & 0 & 1 & 0 & -1 \\
0 & 0 & -1 & 0 & 3 \\
0 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & -1 & 3 \\
-1 & 0 & 1 & 0 & 1 \\
\end{bmatrix}
\quad \rightarrow \quad
\begin{bmatrix}
0 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 0 & -1 \\
1 & -1 & 0 & 0 & 0 & 3 \\
1 & 0 & 1 & 0 & 0 & -1 \\
1 & 0 & -1 & 0 & 0 & 3 \\
1 & 0 & 0 & 1 & 0 & -1 \\
1 & 0 & 0 & -1 & 0 & 3 \\
1 & 0 & 0 & 0 & 1 & -1 \\
1 & 0 & 0 & 0 & -1 & 3 \\
1 & -1 & 0 & 1 & 0 & 1 \\
\end{bmatrix}
\]
Get dependence polyhedral and build DDG

The example loop

```c
for ( i = 1; i <= 3; ++i ) {
    S1: a[i] = i;
    for ( j = 1; j <= 3; ++j )
        S2: b[j] = (b[j] + a[i])/2;
}
```

\[
\mathcal{D}_{S_1 \delta S_2, 1, <a[i_{S_1}], a[i_{S_2}]>}
\]

\[
\mathcal{D}_{S_2 \delta S_2, 1, <b[j_{S_2'}], b[j_{S_2}]>}
\]
PoCC: the Polyhedral Compiler Collection
http://www.cse.ohio-state.edu/~pouchet/software/pocc/

Tools within this package:
Candl (dependence analysis)
Clan (IR extraction)
PIPLib (Parametric Integer Programming)
Pluto (Scheduling and Tiling)

Example: matmul program
At ilab machines:
/ilab/users/zz124/cs671_2013/poly/candl_examples
End of Lecture 6